Práctica 2.3. Procesos

Objetivos

En esta práctica se revisan las funciones del sistema básicas para la gestión de procesos: políticas de planificación, creación de procesos, grupos de procesos, sesiones, recursos de un proceso y gestión de señales.

Contenidos

Preparación del entorno para la práctica
Políticas de planificación
Grupos de procesos y sesiones
Ejecución de programas
Señales

Preparación del entorno para la práctica

Algunos de los ejercicios de esta práctica requieren permisos de superusuario para poder fijar algunos atributos de un proceso, ej. políticas de tiempo real. Por este motivo, es recomendable realizarla en una **máquina virtual** en lugar de las máquinas físicas del laboratorio.

Políticas de planificación

En esta sección estudiaremos los parámetros del planificador de Linux que permiten variar y consultar la prioridad de un proceso. Veremos tanto la interfaz del sistema como algunos comandos importantes.

Ejercicio 1. La política de planificación y la prioridad de un proceso puede consultarse y modificarse con el comando `chrt`. Adicionalmente, los comandos `nice` y `renice` permiten ajustar el valor de *nice* de un proceso. Consultar la página de manual de ambos comandos y comprobar su funcionamiento cambiando el valor de *nice* de la *shell* a -10 y después cambiando su política de planificación a `SCHED_FIFO` con prioridad 12.

```
man chrt-f(fifo) -r  (roundrobin) -o(other) -p(sobre el pid) -v(consultar status)
Man nice renice cambiar prioridad y ejecutar programa
sudo renice -n -10 $$
2049 (process ID) old priority 0, new priority -10
sudo chrt -f -p $$
chrt -p $$
pid 2049's current scheduling policy: SCHED_FIFO
pid 2049's current scheduling priority: 12
```

Ejercicio 2. Escribir un programa que muestre la política de planificación (como cadena) y la prioridad del proceso actual, además de mostrar los valores máximo y mínimo de la prioridad para la política de planificación.

```
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
int main(int argc, char **argv) {
```

```
    int sched_policy = sched_getscheduler(0);
    switch (sched_policy) {
        case SCHED_OTHER:
            printf("SCHED_OTHER ");
        break;
        case SCHED_FIFO:
            printf("SCHED_FIFO ");
        break;
        case SCHED_RR:
            printf("SCHED_RR ");
        break;
        default:
            printf("scheduler: error\n");
        break;
    }
    struct sched_param p;
    sched_getparam(0,&p);
    int max = sched_get_priority_max(sched_policy);
    int min = sched_get_priority_min(sched_policy);
    printf("act: %i, max: %i  min: %i\n", p.sched_priority, max, min);
}
```

Ejercicio 3. Ejecutar el programa anterior en una *shell* con prioridad 12 y política de planificación SCHED_FIFO como la del ejercicio 1. ¿Cuál es la prioridad en este caso del programa? ¿Se heredan los atributos de planificación?.

```
sudo chrt -f-p 12 $$
gcc ej2.c
./a.out
SCHED_FIFO act: 12, max: 99  min: 1
```

Grupos de procesos y sesiones

Los grupos de procesos y sesiones simplifican la gestión que realiza la *shell*, ya que permite enviar de forma efectiva señales a un grupo de procesos (suspender, reanudar, terminar…). En esta sección veremos esta relación y estudiaremos el interfaz del sistema para controlarla.

Ejercicio 4. El comando ps es de especial importancia para ver los procesos del sistema y su estado. Estudiar la página de manual y:

- Mostrar todos los procesos del usuario actual en formato extendido.
- Mostrar los procesos del sistema, incluyendo el identificador del proceso, el identificador del grupo de procesos, el identificador de sesión, el estado y la línea de comandos.
- Observar el identificador de proceso, grupo de procesos y sesión de los procesos. ¿Qué identificadores comparten la *shell* y los programas que se ejecutan en ella? ¿Cuál es el identificador de grupo de procesos cuando se crea un nuevo proceso?

```
man ps
ps -u $USER -f
UID        PID  PPID  C STIME TTY          TIME CMD
cursore+   1370   1365   0 11:13 ?          00:00:00 /usr/bin/openbox --startup
/usr/libexec/openbox-autostart OPENBOX
```

```
cursore+   1379       1   0  11:13  ?           00:00:00 dbus-launch --sh-syntax
--exit-with-session
cursore+   1380      1   0 11:13 ?          00:00:00 /usr/bin/dbus-daemon --fork
--print-pid 5 --print-address 7 --session
cursore+  1448     1  0 11:13 ?        00:00:00 /usr/libexec/imsettings-daemon
cursore+  1452     1  0 11:13 ?        00:00:00 /usr/libexec/gvfsd
cursore+   1457       1   0  11:13  ?           00:00:00 /usr/libexec/gvfsd-fuse
/run/user/1000/gvfs -f -o big_writes
cursore+  1548     1  0 11:13 ?        00:00:00 /usr/bin/VBoxClient --clipboard
cursore+  1550  1548  0 11:13 ?        00:00:00 /usr/bin/VBoxClient --clipboard
cursore+  1559     1  0 11:13 ?        00:00:00 /usr/bin/VBoxClient --display
cursore+  1560  1559  0 11:13 ?        00:00:00 /usr/bin/VBoxClient --display
cursore+  1564     1  0 11:13 ?        00:00:00 /usr/bin/VBoxClient --seamless
cursore+  1566  1564  0 11:13 ?        00:00:00 /usr/bin/VBoxClient --seamless
cursore+  1569     1  0 11:13 ?        00:00:00 /usr/bin/VBoxClient --draganddrop
cursore+  1572  1569  0 11:13 ?        00:00:04 /usr/bin/VBoxClient --draganddrop
cursore+   1582   1370   0 11:13 ?          00:00:00 /usr/bin/ssh-agent /bin/sh -c
exec -l /bin/bash -c "/usr/bin/openbox-session"
cursore+  1605     1  0 11:13 ?        00:00:00 tint2
cursore+    1652          1    0  11:14  ?              00:00:00  /usr/bin/python
/usr/share/system-config-printer/applet.py
cursore+  1653     1  0 11:14 ?        00:00:00 nm-applet
cursore+  1655     1  0 11:14 ?        00:00:00 abrt-applet
cursore+  1835     1  0 11:14 ?        00:00:00 /usr/libexec/dconf-service
cursore+  1911     1  0 11:14 ?        00:00:00 /usr/libexec/at-spi-bus-launcher
cursore+    1923    1911    0  11:14  ?               00:00:00  /bin/dbus-daemon
--config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --pri
cursore+   1964      1   0 11:14 ?         00:00:00 /usr/libexec/at-spi2-registryd
--use-gnome-session
cursore+   1992      1   0 11:14 ?          00:00:00 /usr/bin/pulseaudio --start
--log-target=syslog
cursore+      2015              1      0   11:14   ?                     00:00:03
/usr/libexec/gnome-terminal-server
cursore+  2020     1  0 11:14 ?        00:00:00 /usr/libexec/xdg-desktop-portal
cursore+  2025     1  0 11:14 ?        00:00:00 /usr/libexec/xdg-document-portal
cursore+  2029     1  0 11:14 ?        00:00:00 /usr/libexec/xdg-permission-store
cursore+      2041              1      0   11:14   ?                     00:00:00
/usr/libexec/xdg-desktop-portal-gtk
cursore+  2048  2015  0 11:14 ?        00:00:00 gnome-pty-helper
cursore+  2049  2015  0 11:14 pts/0    00:00:00 bash
cursore+    2393    1605    1  11:23  ?             00:00:21  /usr/share/code/code
--unity-launch
cursore+    2396    2393    0  11:23  ?             00:00:00  /usr/share/code/code
--type=zygote --no-sandbox
cursore+    2554    2396    0  11:23  ?             00:00:00  /usr/share/code/code
--type=renderer --js-flags=--nolazy --disable-mojo-local-storage --no-s
cursore+      2575              1      0   11:24   ?                     00:00:00
/usr/libexec/gvfs-udisks2-volume-monitor
cursore+      2580              1      0   11:24   ?                     00:00:00
/usr/libexec/gvfs-afc-volume-monitor
cursore+      2586              1      0   11:24   ?                     00:00:00
```

```
/usr/libexec/gvfs-gphoto2-volume-monitor
cursore+      2591              1      0   11:24   ?                       00:00:00
/usr/libexec/gvfs-mtp-volume-monitor
cursore+      2596              1      0   11:24   ?                       00:00:00
/usr/libexec/gvfs-goa-volume-monitor
cursore+  2600     1  0 11:24 ?        00:00:00 /usr/libexec/goa-daemon
cursore+  2608     1  0 11:24 ?        00:00:00 /usr/libexec/goa-identity-service
cursore+  2616     1  0 11:24 ?        00:00:00 /usr/libexec/mission-control-5
cursore+   2618        1   0 11:24  ?           00:00:00  /usr/libexec/gvfsd-trash
--spawner :1.3 /org/gtk/gvfs/exec_spaw/0
cursore+   2639       1   0 11:24  ?          00:00:00 /usr/libexec/gvfsd-network
--spawner :1.3 /org/gtk/gvfs/exec_spaw/1
cursore+   2652        1   0 11:24  ?           00:00:00  /usr/libexec/gvfsd-dnssd
--spawner :1.3 /org/gtk/gvfs/exec_spaw/3
cursore+    2674    2396    3  11:24  ?              00:00:43  /usr/share/code/code
--type=renderer --js-flags=--nolazy --disable-mojo-local-storage --no-s
cursore+    2692    2674    0  11:24  ?              00:00:01  /usr/share/code/code
/usr/share/code/resources/app/out/bootstrap --type=extensionHost
cursore+    2693    2674    0  11:24  ?              00:00:00  /usr/share/code/code
/usr/share/code/resources/app/out/bootstrap --type=watcherService
cursore+  3035  2049  0 11:45 pts/0    00:00:00 ps -u cursoredes -f

ps -eo pid,gid,sid,s,command
PID   GID   SID S COMMAND
       1        0        1  S  /usr/lib/systemd/systemd  --switched-root  --system
--deserialize 22
    2     0     0 S [kthreadd]
    3     0     0 S [ksoftirqd/0]
    5     0     0 S [kworker/0:0H]
    6     0     0 S [kworker/u2:0]
    7     0     0 S [migration/0]
    8     0     0 S [rcu_bh]
    9     0     0 S [rcu_sched]
   10     0     0 S [lru-add-drain]
   11     0     0 S [watchdog/0]
   13     0     0 S [kdevtmpfs]
   14     0     0 S [netns]
   15     0     0 S [khungtaskd]
   16     0     0 S [writeback]
   17     0     0 S [kintegrityd]
   18     0     0 S [bioset]
   19     0     0 S [bioset]
   20     0     0 S [bioset]
   21     0     0 S [kblockd]
   22     0     0 S [md]
   23     0     0 S [edac-poller]
   29     0     0 S [kswapd0]
   30     0     0 S [ksmd]
   31     0     0 S [khugepaged]
   32     0     0 S [crypto]
   40     0     0 S [kthrotld]
```

```
 42     0     0 S [kmpath_rdacd]
 43     0     0 S [kaluad]
 44     0     0 S [kpsmoused]
 45     0     0 S [ipv6_addrconf]
 59     0     0 S [deferwq]
 91     0     0 S [kauditd]
268     0     0 S [ata_sff]
272     0     0 S [scsi_eh_0]
274     0     0 S [scsi_tmf_0]
275     0     0 S [scsi_eh_1]
276     0     0 S [scsi_tmf_1]
277     0     0 S [scsi_eh_2]
279     0     0 S [kworker/u2:3]
280     0     0 S [scsi_tmf_2]
311     0     0 S [kworker/0:1H]
352     0     0 S [kdmflush]
353     0     0 S [bioset]
363     0     0 S [kdmflush]
364     0     0 S [bioset]
376     0     0 S [bioset]
377     0     0 S [xfsalloc]
378     0     0 S [xfs_mru_cache]
379     0     0 S [xfs-buf/dm-0]
380     0     0 S [xfs-data/dm-0]
381     0     0 S [xfs-conv/dm-0]
382     0     0 S [xfs-cil/dm-0]
383     0     0 S [xfs-reclaim/dm-]
384     0     0 S [xfs-log/dm-0]
385     0     0 S [xfs-eofblocks/d]
386     0     0 S [xfsaild/dm-0]
457     0   457 S /usr/lib/systemd/systemd-journald
476     0   476 S /usr/sbin/lvmetad -f
490     0   490 S /usr/lib/systemd/systemd-udevd
608     0     0 S [iprt-VBoxWQueue]
697     0     0 S [xfs-buf/sda1]
698     0     0 S [xfs-data/sda1]
699     0     0 S [xfs-conv/sda1]
700     0     0 S [xfs-cil/sda1]
701     0     0 S [xfs-reclaim/sda]
702     0     0 S [xfs-log/sda1]
703     0     0 S [xfs-eofblocks/s]
705     0     0 S [xfsaild/sda1]
706     0     0 S [ttm_swap]
742     0     0 S [rpciod]
743     0     0 S [xprtiod]
745     0   745 S /sbin/auditd
747     0   747 S /sbin/audispd
749     0   747 S /usr/sbin/sedispatch
770     0   770 S /sbin/rngd -f
773     0   773 S /usr/sbin/abrtd -d -s
 774      0    774 S /usr/bin/abrt-watch-log -F BUG: WARNING: at WARNING: CPU:
```

```
INFO: possible recursive locking detected ernel BUG at list_de
   776     81    776 S /usr/bin/dbus-daemon --system --address=systemd: --nofork
--nopidfile --systemd-activation
  782    32   782 S /sbin/rpcbind -w
  784     0   784 S /usr/sbin/gssproxy -D
  793   172   793 S /usr/libexec/rtkit-daemon
  794     0   794 S /usr/sbin/ModemManager
  795   998   795 S /usr/lib/polkit-1/polkitd --no-debug
  801      0   801 S /usr/bin/abrt-watch-log -F Backtrace /var/log/Xorg.0.log --
/usr/bin/abrt-dump-xorg -xD
  812     0   812 S /usr/sbin/smartd -n -q never
  814     0   814 S /usr/libexec/accounts-daemon
  815     0   815 S /usr/lib/systemd/systemd-logind
  816   993   816 S /usr/bin/lsmd -d
  821     0   821 S /usr/libexec/udisks2/udisksd
     823             0        823   S   /usr/sbin/alsactl   -s   -n   19   -c   -E
ALSA_CONFIG_PATH=/etc/alsa/alsactl.conf --initfile=/lib/alsa/init/00main rdaemon
  847     0   847 S /usr/sbin/mcelog --ignorenodev --daemon --syslog
  851   991   850 S /usr/sbin/chronyd
  864     0   825 S /bin/bash /usr/sbin/ksmtuned
 1017     0  1017 S /usr/sbin/sshd -D
 1018     0  1018 S /usr/sbin/cupsd -f
 1022     0  1022 S /usr/bin/python -Es /usr/sbin/tuned -l -P
 1023     0  1023 S /usr/sbin/rsyslogd -n
 1035     0  1035 S /usr/sbin/atd -f
 1036     0  1036 S /usr/sbin/crond -n
 1146     0  1146 S /usr/libexec/postfix/master -w
 1147    89  1146 S pickup -l -t unix -u
 1148    89  1146 S qmgr -l -t unix -u
 1325     0  1325 S /usr/sbin/gdm
 1343     0  1341 S /usr/sbin/VBoxService --pidfile /var/run/vboxadd-service.sh
 1352      0   1352 S /usr/bin/X :0 -background none -noreset -audit 4 -verbose
-auth /run/gdm/auth-for-gdm-6ySYD9/database -seat seat0 -nolis
 1365  1000  1325 S gdm-session-worker [pam/gdm-autologin]
 1370   1000   1370 S /usr/bin/openbox --startup /usr/libexec/openbox-autostart
OPENBOX
 1379  1000  1370 S dbus-launch --sh-syntax --exit-with-session
 1380   1000   1380 S /usr/bin/dbus-daemon --fork --print-pid 5 --print-address 7
--session
 1448  1000  1380 S /usr/libexec/imsettings-daemon
 1452  1000  1380 S /usr/libexec/gvfsd
 1457  1000  1380 S /usr/libexec/gvfsd-fuse /run/user/1000/gvfs -f -o big_writes
 1548  1000  1546 S /usr/bin/VBoxClient --clipboard
 1550  1000  1546 S /usr/bin/VBoxClient --clipboard
 1559  1000  1556 S /usr/bin/VBoxClient --display
 1560  1000  1556 S /usr/bin/VBoxClient --display
 1564  1000  1561 S /usr/bin/VBoxClient --seamless
 1566  1000  1561 S /usr/bin/VBoxClient --seamless
 1569  1000  1567 S /usr/bin/VBoxClient --draganddrop
 1572  1000  1567 S /usr/bin/VBoxClient --draganddrop
  1582    1000    1582  S  /usr/bin/ssh-agent  /bin/sh  -c  exec  -l  /bin/bash  -c
```

```
"/usr/bin/openbox-session"
 1605  1000  1370 S tint2
 1652  1000  1370 S /usr/bin/python /usr/share/system-config-printer/applet.py
 1653  1000  1370 S nm-applet
 1655  1000  1370 S abrt-applet
 1835  1000  1380 S /usr/libexec/dconf-service
 1911  1000  1380 S /usr/libexec/at-spi-bus-launcher
       1923              1000              1380       S       /bin/dbus-daemon
--config-file=/usr/share/defaults/at-spi2/accessibility.conf             --nofork
--print-address 3
 1964  1000  1380 S /usr/libexec/at-spi2-registryd --use-gnome-session
 1992  1000  1991 S /usr/bin/pulseaudio --start --log-target=syslog
 2015  1000  1380 R /usr/libexec/gnome-terminal-server
 2020  1000  1380 S /usr/libexec/xdg-desktop-portal
 2025  1000  1380 S /usr/libexec/xdg-document-portal
 2029  1000  1380 S /usr/libexec/xdg-permission-store
 2041  1000  1380 S /usr/libexec/xdg-desktop-portal-gtk
 2048    22  1380 S gnome-pty-helper
 2049  1000  2049 S bash
 2393  1000  2393 S /usr/share/code/code --unity-launch
 2396  1000  2393 S /usr/share/code/code --type=zygote --no-sandbox
 2554   1000   2393  S  /usr/share/code/code  --type=renderer  --js-flags=--nolazy
--disable-mojo-local-storage --no-sandbox --disable-features=Co
 2575  1000  1380 S /usr/libexec/gvfs-udisks2-volume-monitor
 2580  1000  1380 S /usr/libexec/gvfs-afc-volume-monitor
 2586  1000  1380 S /usr/libexec/gvfs-gphoto2-volume-monitor
 2591  1000  1380 S /usr/libexec/gvfs-mtp-volume-monitor
 2596  1000  1380 S /usr/libexec/gvfs-goa-volume-monitor
 2600  1000  1380 S /usr/libexec/goa-daemon
 2608  1000  1380 S /usr/libexec/goa-identity-service
 2616  1000  1380 S /usr/libexec/mission-control-5
   2618       1000       1380    S    /usr/libexec/gvfsd-trash    --spawner    :1.3
/org/gtk/gvfs/exec_spaw/0
   2639       1000       1380    S    /usr/libexec/gvfsd-network    --spawner    :1.3
/org/gtk/gvfs/exec_spaw/1
   2652       1000       1380    S    /usr/libexec/gvfsd-dnssd    --spawner    :1.3
/org/gtk/gvfs/exec_spaw/3
  2674   1000   2393  S  /usr/share/code/code  --type=renderer  --js-flags=--nolazy
--disable-mojo-local-storage --no-sandbox --disable-features=Co
       2692              1000              2393       S       /usr/share/code/code
/usr/share/code/resources/app/out/bootstrap --type=extensionHost
       2693              1000              2393       S       /usr/share/code/code
/usr/share/code/resources/app/out/bootstrap --type=watcherService
 2751     0     0 R [kworker/0:0]
 2850     0     0 S [kworker/0:1]
 2945     0     0 S [kworker/0:2]
 3106     0   825 S sleep 60
 3118  1000  2049 R ps -eo pid,gid,sid,s,command
```

el sid del nuevo proceso es el pid
el gid 1000

Ejercicio 5. Escribir un programa que muestre los identificadores del proceso: identificador de proceso, de proceso padre, de grupo de procesos y de sesión. Mostrar además el número máximo de ficheros que puede abrir el proceso y el directorio de trabajo actual.

```
int main() {
    printf("pid: %i \n", getpid());
    printf("ppid: %i \n", getppid());
    printf("gid: %i \n", getpgid(getpid()));
    printf("sid: %i \n", getsid(getpid()));
    struct rlimit param;
    if (getrlimit(RLIMIT_NOFILE, &param) == -1) {
        perror("error rlimit");
        return -1;
    }
    printf("Nº maximo de ficheros : %ld\n", param.rlim_max);
    char *path = malloc(sizeof(char)*(4096 + 1));
    char *rpath = getcwd(path, 4096 + 1);
    printf("cwd: %s\n", path);
    free (path);
    return 0;
}

./a.out
pid: 3388
ppid: 2049
gid: 3388
sid: 2049
Nº maximo de ficheros : 4096
cwd: /home/cursoredes/Downloads/jorge
```

Ejercicio 6. Un demonio es un proceso que se ejecuta en segundo plano para proporcionar un servicio. Normalmente, un demonio está en su propia sesión y grupo. Para garantizar que es posible crear la sesión y el grupo, el demonio crea un nuevo proceso para ejecutar la lógica del servicio y crear la nueva sesión. Escribir una plantilla de demonio (creación del nuevo proceso y de la sesión) en el que únicamente se muestren los atributos del proceso (como en el ejercicio anterior). Además, fijar el directorio de trabajo del demonio a `/tmp`.

¿Qué sucede si el proceso padre termina antes que el hijo (observar el PPID del proceso hijo)? ¿Y si el proceso que termina antes es el hijo (observar el estado del proceso hijo con `ps`)?

Nota: Usar `sleep(3)` o `pause(3)` para forzar el orden de finalización deseado.

```
void atributos(char *type){
    printf("%s pid: %i\n", type,getpid());
    printf("%s ppid: %i\n", type, getppid());
    printf("%s gid: %i \n", type, getpgid(getpid()));
    printf("%s sid: %i\n", type, getsid(getpid()));

    struct rlimit param;
    getrlimit(RLIMIT_NOFILE, &param) == -1;
    printf("Nº maximo de ficheros : %ld\n", param.rlim_max);
```

```
    char *path = malloc(sizeof(char)*(4096 + 1));
    char *rpath = getcwd(path, 4096 + 1);
    printf("cwd: %s\n", path);
    free (path);
}
int main() {
    pid_t pid = fork();
    switch (pid) {
    case -1:
        perror("fork");
        exit(1);
    break;
    case 0:
        chdir("/tmp");
        setsid();
        sleep(3);
        printf("HIJO proceso %i (proc_padre: %i)\n",getpid(),getppid());
        atributos("HIJO");
        break;
    default:
        printf("PADRE proceso %i (proc_padre: %i)\n",getpid(),getppid());
        atributos("PADRE");
        break;
    }
    return 0;
}

gcc ej6.c
./a.out
PADRE proceso 3711 (proc_padre: 2049)
PADRE pid: 3711
PADRE ppid: 2049
PADRE gid: 3711
PADRE sid: 2049
Nº maximo de ficheros : 4096
cwd: /home/cursoredes/Downloads/jorge
HIJO proceso 3712 (proc_padre: 1)
HIJO pid: 3712
HIJO ppid: 1
HIJO gid: 3712
HIJO sid: 3712
Nº maximo de ficheros : 4096
cwd: /tmp

Si el padre termina antes el hijo huérfano lo recoge la shell
```

Ejecución de programas

Ejercicio 7. Escribir dos versiones, una con `system(3)` y otra con `execvp(3)`, de un programa que ejecute otro programa que se pasará como argumento por línea de comandos. En cada caso, se

debe imprimir la cadena “`El comando terminó de ejecutarse`” después de la ejecución. ¿En qué casos se imprime la cadena? ¿Por qué?

Nota: Considerar cómo deben pasarse los argumentos en cada caso para que sea sencilla la implementación. Por ejemplo: ¿qué diferencia hay entre `./ej7 ps -el` y `./ej7 “ps -el”`?

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv){
    if (argc < 2) {
        printf("error comando\n");
        return -1;
    }
    if (execvp(argv[1], argv + 1) == -1) {
        printf("error ejecución\n");
    }
    printf("El comando terminó de ejecutarse\n");
    return 0;
}
int main(int argc, char **argv){

    if (argc < 2) {
        printf("error comando\n");
        return -1;
    }
    int longitud = 1;
    int i;
    for (i = 1; i < argc; i++)
        longitud += strlen(argv[i]) + 1;
    char *comando = malloc(sizeof(char)*longitud);
    strcpy(comando, "");
    for (i = 1; i < argc; i++) {
        strcat(comando, argv[i]);
        strcat(comando, " ");
    }
    if (system(comando) == -1) {
        printf("error ejecución\n");
    }
    printf("El comando terminó de ejecutarse\n");
    return 0;
}
El comando terminó de ejecutarse se muestra en system y en exec en caso de error ejecución
Las comillas hacen funcionar como un único parámetro y nos ahorraría unir los parámetros para system
```

Ejercicio 8. Usando la versión con `execvp(3)` del ejercicio 7 y la plantilla de demonio del ejercicio 6, escribir un programa que ejecute cualquier programa como si fuera un demonio. Además, redirigir los flujos estándar asociados al terminal usando `dup2(2)`:

- La salida estándar al fichero `/tmp/daemon.out`.
- La salida de error estándar al fichero `/tmp/daemon.err`.
- La entrada estándar a `/dev/null`.

Comprobar que el proceso sigue en ejecución tras cerrar la *shell*.

```
int main(int argc, char **argv){
    if (argc < 2) {
        printf("ERROR: Introduce el comando.\n");
        return -1;
    }
    pid_t pid = fork();
    switch (pid) {
        case -1:
            perror("fork");
            exit(-1);
        break;
        case 0:;
            setsid();
            printf("Hijo proceso %i (proc_padre: %i)\n",getpid(),getppid());
            int fd_est = open("/tmp/daemon.out",O_CREAT | O_RDWR, 00777);
            int fd_err = open("/tmp/daemon.err", O_CREAT | O_RDWR, 00777);
            int fd_ent = open("/dev/null", O_CREAT | O_RDWR, 00777);
            int estandar = dup2(fd_est,2);
            int err = dup2(fd_err, 1);
            int null = dup2(fd_ent, 0);
            if (execvp(argv[1], argv + 1) == -1) {
                printf("error ejecucion\n");
                exit(-1);
            }
        break;
        default:
            printf("Padre proceso %i (proc_padre: %i)\n",getpid(),getppid());
        break;
    }
    return 0;
}
```

Señales

Ejercicio 9. El comando `kill(1)` permite enviar señales a un proceso o grupo de procesos por su identificador (`pkill` permite hacerlo por nombre de proceso). Estudiar la página de manual del comando y las señales que se pueden enviar a un proceso.

```
man 1 kill
kill -l
 1) SIGHUP    2) SIGINT    3) SIGQUIT   4) SIGILL    5) SIGTRAP
 6) SIGABRT   7) SIGBUS    8) SIGFPE    9) SIGKILL  10) SIGUSR1
11) SIGSEGV  12) SIGUSR2  13) SIGPIPE  14) SIGALRM  15) SIGTERM
16) SIGSTKFLT17) SIGCHLD  18) SIGCONT  19) SIGSTOP  20) SIGTSTP
21) SIGTTIN  22) SIGTTOU  23) SIGURG   24) SIGXCPU  25) SIGXFSZ
26) SIGVTALRM27) SIGPROF  28) SIGWINCH 29) SIGIO    30) SIGPWR
31) SIGSYS   34) SIGRTMIN 35) SIGRTMIN+1      36) SIGRTMIN+2      37) SIGRTMIN+3
```

```
38) SIGRTMIN+4      39) SIGRTMIN+5      40) SIGRTMIN+6      41) SIGRTMIN+7
42) SIGRTMIN+8
43) SIGRTMIN+9      44) SIGRTMIN+10     45) SIGRTMIN+11     46) SIGRTMIN+12
47) SIGRTMIN+13
48) SIGRTMIN+14     49) SIGRTMIN+15     50) SIGRTMAX-14     51) SIGRTMAX-13
52) SIGRTMAX-12
53) SIGRTMAX-11     54) SIGRTMAX-10     55) SIGRTMAX-9      56) SIGRTMAX-8
57) SIGRTMAX-7
58) SIGRTMAX-6      59) SIGRTMAX-5      60) SIGRTMAX-4      61) SIGRTMAX-3
62) SIGRTMAX-2
63) SIGRTMAX-1      64) SIGRTMAX
```

Ejercicio 10. En un terminal, arrancar un proceso de larga duración (ej. `sleep 600`). En otra terminal, enviar diferentes señales al proceso, comprobar el comportamiento. Observar el código de salida del proceso. ¿Qué relación hay con la señal enviada?

```
kill 4274
Terminated
SIGHUP : hangup
SIGINT: (nada)
SIGQUIT : Quit (core dumped)
SIGILL : Illegal instruction (core dumped)
SIGTRAP: Trace/breakpoint trap (core dumped)
SIGKILL: killed
```

Ejercicio 11. Escribir un programa que bloquee las señales `SIGINT` y `SIGTSTP`. Después de bloquearlas el programa debe suspender su ejecución con `sleep(3)` un número de segundos que se obtendrán de la variable de entorno `SLEEP_SECS`. Al despertar, el proceso debe informar de si recibió la señal `SIGINT` y/o `SIGTSTP`. En este último caso, debe desbloquearla con lo que el proceso se detendrá y podrá ser reanudado en la *shell* (imprimir una cadena antes de finalizar el programa para comprobar este comportamiento).

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
int main(int argc, char **argv) {
      if (argc != 2) {
        printf("error parametros\n");
        return -1;
    }
    sigset_t set;
    sigemptyset(&set);
    sigaddset(&set, SIGINT);
    sigaddset(&set, SIGTSTP);
    sigprocmask(SIG_BLOCK, &set, NULL);
    char *sleep_secs = getenv("SLEEP_SECS");
    int seconds = atoi(sleep_secs);
    printf("Sleep for %d seconds\n", seconds);
    sleep(seconds);
    sigset_t pending;
    sigpending(&pending);
    if (sigismember(&pending, SIGINT) == 1) {
```

```
        printf("SIGINT received\n");
        sigdelset(&set, SIGINT);
    }
    else {
        printf("SIGINT wasnt received\n");
    }
    if (sigismember(&pending, SIGTSTP) == 1) {
        printf("SIGTSTP received\n");
        sigdelset(&set, SIGTSTP);
    }
    else {
        printf("SIGTSTP wasnt received\n");
    }
    sigprocmask(SIG_UNBLOCK, &set, NULL);
    printf("end\n");
    return 0;
}
```

Ejercicio 12. Escribir un programa que instale un manejador para las señales `SIGINT` y `SIGTSTP`. El manejador debe contar las veces que ha recibido cada señal. El programa principal permanecerá en un bucle que se detendrá cuando se hayan recibido 10 señales. El número de señales de cada tipo se mostrará al finalizar el programa.

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

volatile int int_count = 0;
volatile int tstp_count = 0;
void hler(int signal){
    if (signal == SIGINT)
        int_count++;
    if (signal == SIGTSTP)
        tstp_count++;
}
int main(){
    struct sigaction act;
    sigaction(SIGINT, NULL, &act);
    act.sa_handler = hler;
    sigaction(SIGINT, &act, NULL);
    sigaction(SIGTSTP, NULL, &act);
    act.sa_handler = hler;
    sigaction(SIGTSTP, &act, NULL);
    sigset_t set;
    sigemptyset(&set);
    while (int_count + tstp_count < 10)
        sigsuspend(&set);
    printf("SIGINT recibida %i veces\n", int_count);
    printf("SIGTSTP recibida %i veces\n", tstp_count);
    return 0;
}
```

```
/a.out 10
[cursoredes@localhost jorge]$ kill -2 5496
[cursoredes@localhost jorge]$ kill -2 5496
[cursoredes@localhost jorge]$ kill -2 5496
[cursoredes@localhost jorge]$ kill -20 5496
[cursoredes@localhost jorge]$ kill -20 5496
[cursoredes@localhost jorge]$ kill -20 5496
[cursoredes@localhost jorge]$ kill -20 5496
[cursoredes@localhost jorge]$ kill -20 5496
[cursoredes@localhost jorge]$ kill -20 5496
[cursoredes@localhost jorge]$ kill -20 5496
SIGINT recibida 4 veces
SIGTSTP recibida 6 veces
```

Ejercicio 13. Escribir un programa que realice el borrado programado del propio ejecutable. El programa tendrá como argumento el número de segundos que esperará antes de borrar el fichero. El borrado del fichero se podrá detener si se recibe la señal `SIGUSR1`

Nota: Usar `sigsuspend(2)` para suspender el proceso y la llamada al sistema apropiada para borrar el fichero.

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

volatile int stop = 0;
void hler(int signal){
  if (signal == SIGUSR1) stop = 1;
}
int main(int argc, char **argv) {
    if (argc != 2) {
        printf("error parametro\n");
        return -1;
    }
    sigset_t set;
    sigemptyset(&set);
    sigaddset(&set, SIGUSR1);
    sigprocmask(SIG_UNBLOCK, &set, NULL);
    struct sigaction act;
    sigaction(SIGUSR1, NULL, &act);
    act.sa_handler = hler;
    sigaction(SIGUSR1, &act, NULL);
    int seconds = atoi(argv[1]);
    int i = 0;
    while (i < seconds && stop == 0) {
        i++;
        sleep(1);
    }
    if (stop == 0) {
        printf("Se borra el ejecutable por haber pasado %i segundos\n", seconds);
        unlink(argv[0]);
```

```
    }
    else
        printf("Se recibe la señal SIGUSR1 y no se borra el ejecutable\n");
    return 0;
}
./a.out 100
kill -10 5665
Se recibe la señal SIGUSR1 y no se borra el ejecutable
/a.out 10
Se borra el ejecutable por haber pasado 10 segundos
```